An Applications Approach to Evolvable Hardware

Reid Porter and Kevin McCabe
Los Alamos National Laboratory,
Space and Remote Sensing,
Los Alamos, New Mexico USA 87545.
rporter, kmccabe@lanl.gov

Neil Bergmann
Cooperative Research Centre for Satellite Systems ,
Queensland University of Technology,
Brisbane Australia 40001.
n.bergman@qut.edu.au

Abstract

We discuss the use of Field Programmable Gate Arrays (FPGAs) as hardware accelerators in genetic algorithm (GA) applications. The research is particularly focused on image processing optimization problems where fitness evaluation is computationally demanding and poorly suited to micro-processor systems. This research identifies key design principles for FPGA based GA and suggests a novel 2 stage reconfiguration technique. We demonstrate its effectiveness in obtaining significant speed-up; and illustrate the unique hardware GA design environment where representation is driven by a combination of hardware architecture and problem domain.

1. Introduction

Evolvable hardware attempts to apply evolutionary principles to the design of electronic circuits. We are also interested in using genetic algorithms to find suitable hardware circuits for particular applications but are motivated by the long execution times of software GA experiments. GAs are an effective optimization procedure which use a large number of candidate solutions (population) to converge over time to a global optimum [2]. Maintaining such a population leads to robust solutions in many problem domains but also leads to large computation times. This is particularly true when GAs are applied to image processing problems which are generally poorly suited to traditional sequential processors. Hardware acceleration of GAs is a difficult problem due to the application specific nature of representation, and fitness evaluation. FPGAs are a flexible implementation alternative that can provide the application specific architecture necessary for GA speed-up while still maintaining software flexibility.

We suggest FPGA based GA-accelerators using a novel 2-stage reconfiguration technique. A GA-accelerator needs to be configurable for (1) different problems and (2) different candidate solutions (also referred to as individuals) of the population for each problem. For each problem the FPGA is configured once at the start of a GA run using the FPGA programming bit-stream. This method is too slow to configure each individual of the GA population and so we introduce a second level of configurability using control lines to dictate arrangement of a fixed set of operators specific to that problem.

Section 2 introduces the two stage reconfiguration technique and associated terminology. The first and second levels of configurability are described in more detail in Sections 3 and 4 respectively. Section 5 describes the application of the technique to a multi-spectral feature identification problem and provides results from initial experiments.

The key to efficient implementation of the 2-stage reconfiguration technique is in the reuse of hardware resources from one individual to the next. This maximizes the amount of FPGA area being used at any one time leading to increased performance. We describe three levels of hardware reuse in this paper: gate level reuse is the most commonly used method in evolvable hardware systems and is described in Section 3, arithmetic level reuse is described in Section 4 and application specific operator level reuse in Section 5.1.

2. The GA Isostructure

Usually the most computationally intensive part of a GA is the fitness evaluation. This involves evaluating how well each individual in a GA population solves the particular problem. When GAs are applied to image processing problems this can involve several data intensive image processing operations for each individual. If each individual can be implemented in hardware, this fitness evaluation can be carried out at high speed, reducing GA run times considerably [5]. A problem with high density FPGAs is that reconfiguration time can often become large compared to the time required for fitness evaluation. To avoid this problem we suggest that two levels of configuration are required.

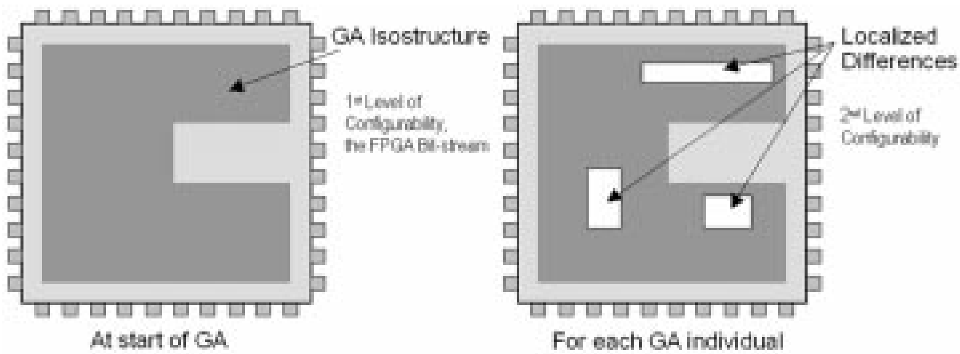

Figure 1. Two stage reconfiguration

In most GA experiments, individuals have similar implementation requirements. A key to efficient implementation is to localize where GA individuals differ so that reconfiguration time can be minimized. We present Figure 1 and the following terminology for clarity. The GA isostructure is implemented with the first level of configurability, the FPGA programming bit-stream, and is common to all individuals in a GA run. The GA isostructure requires structure to obtain significant speed-up, and also flexibility to implement all GA individuals of interest. The second level of configurability is then used to rapidly fine tune the isostructure, by setting control lines, to implement particular GA individuals. The result of this two stage configuration process is a hardware implementation of a particular GA individual where fitness evaluation can be carried out at high speed.

The choice of isostructure is an important one and is dictated by a combination of application domain and hardware resources. It leads to unique hardware search spaces that will have direct effect upon the evolvability of the system. This will be discussed further in Section 5.2.

3. The First Level of Configurability

With rapidly reconfigurable FPGAs such as the Xilinx XC6000 series devices only the first level of configurability is required. FPGA programming is generally still a two step process. (1) An isostructure is first implemented at the start of the GA run and (2) particular GA individuals are then configured as required using partial reconfiguration. Since the GA is operating on the programming bit-stream directly, the functionality and connectivity of each FPGA logic cell can vary from one GA individual to the next. We consider this as hardware reuse at the gate level.

We investigated this feature by evolving cellular automata (CA) rule tables to perform basic pattern recognition. Papers by Mitchell [8] and Sahota [10] describe similar experiments performed in software. Our experiment, described in detail in [9], implemented 5 variable, 2 state CA on a XC6000 series FPGA using combinatorial logic trees. The logic tree isostructure is illustrated in Figure 2 and is configured by selecting inputs to the logic gates and multiplexers as well as choosing suitable functions for the logic gates. This isostructure made a high speed implementation possible while still providing flexibility to implement all GA individuals of interest, in this case a 5 variable CA rule table.

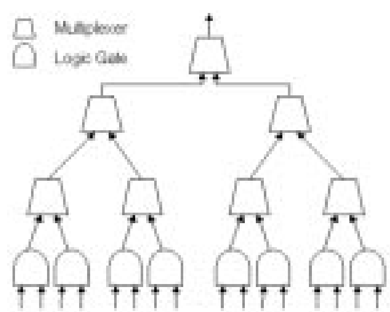

Figure 2. Logic Tree Isostructure

Representation based on the logic tree isostructure was compared against a software based CA rule table GA. It was found that the logic tree isostructure could implement problem specific constraints more easily than CA rule table representations leading to improved performance. Evaluating the fitness of the hardware individual was 38 times faster than software fitness evaluation.

4. A Second Level of Configurability

FPGA manufacturers have moved away from rapidly reconfigurable architectures in favor of high density devices. Such devices suffer from long reconfiguration times and therefore the second level of configurability is required. The FPGA bit-steam is suitable for configuring the GA isostructure since this is necessary only at the start of a GA run. The second level of configurability must be incorporated into the GA isostructure directly using multiplexers and control lines. Hardware reuse is an important design consideration at this stage since incorporating the control lines can quickly dominate hardware resources. Most applications do not require the fine grain flexibility of gate level configurability and therefore we adopt a problem orientated approach to hardware reuse.

Generalized pipelined arrays, suggested by Kamal in [4] are an excellent example of how the second level of configurability can be implemented. These arrays can be configured by setting control lines to perform a number of common operations such as multiplication, division, square root, and squaring. These operations have very similar hardware requirements and therefore can be combined with large area cost savings. An arithmetic array was implemented on Altera Flex10K devices and details are summarized in Table 1. Through intelligent reuse of hardware

Operator	$\times$	$\div$	$\surd$	$\times \div \surd$
Estimated Area Cost (Logic Cells)	206	286	305	429
Resource Gain				1.98

Table 1. Area Cost Estimates

the computational resources available to the GA is effectively doubled. We consider this as hardware reuse at the arithmetic level and suggest such arrays as useful building blocks for many GA applications.

5. Application to Multi-Spectral Feature Identification

Multi-spectral image processing is a data intensive and time consuming process traditionally employing groups of analysts to manually identify regions of interest within a particular data set. Automatic feature identification algorithms have been developed but are often application and data set specific and lack generality to be applied to the wide range of problems analysts may encounter. We attempt to use GAs to automatically generate and fine tune feature identification algorithms for particular applications or data sets that have not yet been considered. We are particularly interested in area-based features that are used in terrain classification and land use characterization.

Our approach is closely related to Genetic Programming methodologies where a number of image processing operators are combined with the multi-spectral input channels to produce executable algorithms. The choice of operators is an important one and is dictated largely by hand crafted classification algorithms developed in the remote sensing community.

5.1. Hardware Reuse at the Operator Level

Operators are grouped according to their hardware requirements and are summarized in Table 2.

Type	Examples
Spectral	NDVI, Linear Scale, Linear Combination
Spatial	Mean, Standard Deviation, Convolution, Rank Order
Threshold	Clipping, Boolean, Band Pass

Table 2. Types of Operators

Spectral operators are used extensively in the remote sensing community and are applied on a pixel by pixel basis to one or more input channels. Their data requirements suggest a pipelined architecture and we therefore use Kamal's generalized arithmetic pipeline as a starting point. A powerful measure of land type in remote sensing is the Normalized Differential Vegetation Index (NDVI) calculated as: $NDVI = \frac{A-B}{A+B}$ where A and B represent two multispectral input channels [6]. We use this problem specific knowledge to define the configurable spectral operator architecture of Figure 3.

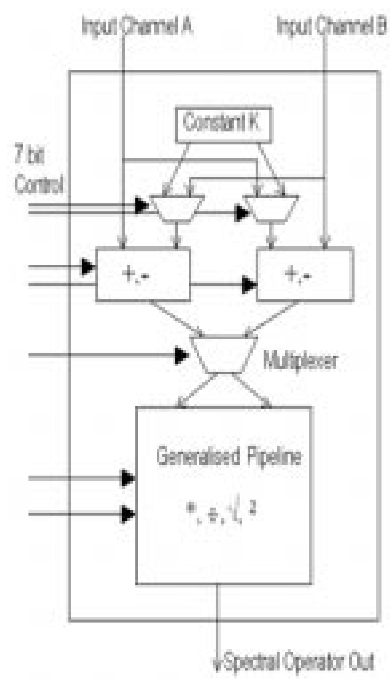

Figure 3. Programmable Spectral Operator

With the increasing spatial resolution of multi-spectral sensors, texture information becomes increasingly useful [3]. Spatial operators are applied to a local neighborhood of pixels taken from a single input channel. The predominate hardware cost for spatial operators in a pipelined architecture is accessing the local neighborhood and this can be shared between different operators. Thresholding operators also have very similar hardware requirements and can share resources easily. By combining operators into larger, programmable macro operators we can employ hardware reuse at the operator level.

5.2. Choice of Isostructure

A GA individual will contain a number of operators in a specific order. One individual may use a spectral operator followed by a spatial operator, while another will use a spatial operator followed by a spectral operator. In software the GA can be used to find the optimum order of operators. In hardware some structure must be imposed on the search space in order to make implementation possible. The effect of GA isostructure on evolvability is not clear and knowledge of the problem domain plays an important role. Miller in [7] discusses similar issues of evolvability in evolving 2-bit binary multipliers. Adaptive representation schemes have been suggested [1], which may be applicable to this problem. Since isostructural optimisation would only need to be performed once at the design stage, efficiency gains may be possible if the problem domain has general structural properties.

5.3. Summary of Initial Experiments

The isostructure used in our initial experiments is illustrated in Figure 4. The GA selects the input channels to the top level spectral and spatial operators, as well as the precise functionality of the 5 image operators. Figure 5 is an example of the training data used showing a visible band of the multi-spectral input (top) and desired output image (bottom). The required area feature, in this case a type of vegetation, was identified by hand and used as binary truth.

Experiments used a population of 100 and ran for 30 generations using an elitest strategy similar to that used in [8]. Crossover and mutation were constrained to maintain the GA isostructure. We compared the performance over 30 runs of the structured GA to a software version where operator order was not specified. Both types of GA performed equally well with an isostructural individual achieving the highest overall accuracy of 95.7%.

There was not enough statistical difference in the results to discern the better GA strategy, but we are encouraged by the similar performance and conclude the GA isostructure has the necessary flexibility to implement most individuals of interest. Further investigations are required including

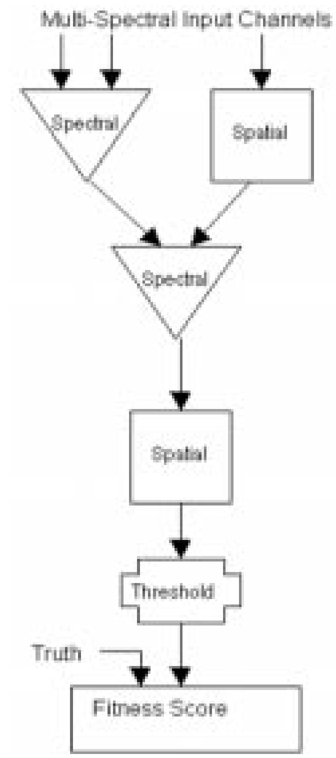

Figure 4. GA isostructure

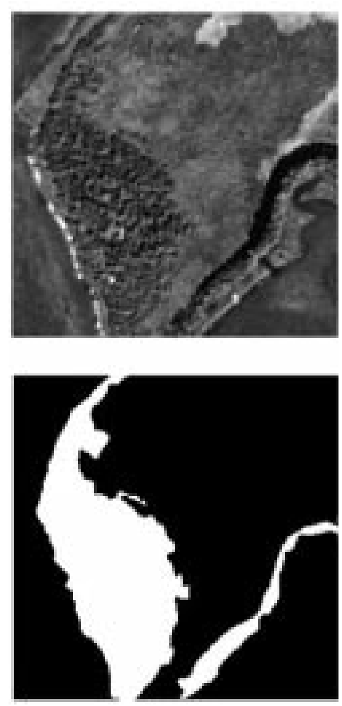

Figure 5. Example training images

Operator	#Logic Cells (12160 Total)	% of FPGA
Spectral	650 * 2	11
Spatial	1230 * 2	20
Threshold	125	1
Total	3885	32
On Chip Memory	8Kbits out of 40Kbits	20

Table 3. Area Cost Estimates

variation of the GA isostructure. We believe an isostructural GA is possible that will include all GA individuals of interest and may even out-perform unconstrained GA runs due to the reduced search space.

We have estimated the FPGA area requirements for the GA isostructure of Figure 4 which are summarized in Table 3. The design is targeted at an Altera Flex10K250 device and will be implemented in the near future. The isostructure uses one third of the FPGA resources indicating more complicated structures are possible. Future experiments will look at making use of these resources as we learn more about the problem with new operators, advanced classification techniques and intelligent fitness case selection. The target clock speed of the GA isostructure is 40Mhz which achieves a speed-up of 50-100 times an optimized Pentium-II, 450 MHz implementation.

6. Conclusion

Some key concepts in the implementation of FPGA based GA-accelerators have been identified. These include a two stage approach to reconfiguration made necessary by long reconfiguration times of high density FPGA devices. The first level of configurability, the FPGA bit-stream, is used to implement a structure common to all GA individuals at the start of a GA run. This structure is referred to as the GA isostructure and is fine tuned with the second level of configurability to implement all GA individuals of interest. This second level of configurability must be incorporated into the GA isostructure directly by using control lines and multiplexers.

We identified the importance of hardware reuse in implementing the second level of configurability and suggested a problem orientated approach. Reuse at several levels was discussed and arithmetic building blocks suggested. We discussed the importance of GA isostructure, dictated by a combination of problem domain and hardware resources, on evolvability and described initial experiments in multispectral feature identification.

References

[1] L. Altenberg. Evolving better representations through selective genome growth. *Proceedings 1st I.E.E.E. Conference on Evolutionary Computation*, June 1994.

[2] D. Golberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[3] R. Haralick and K. Shanmugam. Combined spectral and spatial processing of erts imagery data. *Remote Sensing of Environment*, 3:3–13, 1974.

[4] A. K. Kamal, H. Singh, and D. Agrawal. A generalized pipeline array. *I.E.E.E. Transactions on Computers*, C-23:533–536, May 1974.

[5] J. R. Koza et. al. Rapid reconfigurable field-programmable gate arrays for accelerating fitness evaluation in genetic programming. *Late Breaking Papers at the Genetic Programming 1997 Conference*, pages 121–131, 1997.

[6] C. Leprieur, M. Verstraete, and B. Pinty. Evaluation of the performance of various vergetation indices to retrieve vegetation cover from avhrr data. *Remote Sensing Reviews*, 10:265–284, 1994.

[7] J. Miller and P. Thomsom. Aspects of digital evolution: Evolvability and architecture. *Parallel Problem Solving from Nature - PPSN V. 5th International Conference*, September 1998.

[8] M. Mitchell, J. P. Crutchfield, and R. Das. Evolving cellular automata with genetic algorithms: A review of recent work. *First International Conference on Evolutionary Computation and Its Applications (EvCA'96)*, 1996.

[9] R. Porter and N. Bergmann. Evolving fpga based cellular automata. *SEAL'98 : Simulated Evolution and Learning*, October 1998.

[10] P. Sahota, M. F. Daemi, and D. G. Elliman. Training genetically evolving cellular automata for image processing. *International Symposium on Speech, Image Processing and Neural Networks*, April, 1994.